

Gradient Coding

Rashish Tandon, Qi Lei, Alexandros G. Dimakis

The University of Texas at Austin
Austin, TX

{rashish, leiqi, dimakis}@{cs, ices, austin}.utexas.edu

Nikos Karampatziakis

Microsoft Research
Redmond, WA

nikosk@microsoft.com

Abstract

We propose a novel coding theoretic framework for mitigating stragglers in distributed learning. We show how carefully replicating data blocks and coding across gradients can provide tolerance to failures and stragglers for synchronous Gradient Descent. We implement our scheme in MPI and show how we compare against baseline architectures in running time and generalization error.

1 Introduction

We propose a novel coding theoretic framework for mitigating stragglers in distributed learning. The central idea can be seen through the simple example of Figure 1: Consider synchronous Gradient Descent (GD) on three workers (W_1, W_2, W_3). The baseline vanilla system is shown on the left figure and operates as follows: The three workers have different partitions of the labeled data stored locally (D_1, D_2, D_3) and all share the current model. Worker 1 computes the gradient of the model on examples in partition D_1 , denoted by g_1 . Similarly, Workers 2 and 3 compute g_2 and g_3 . The three gradient vectors are then communicated to a central node (called the master/aggregator) A which computes the full gradient by summing these vectors $g_1 + g_2 + g_3$ and updates the model with a gradient step. The new model is then sent to the workers and the system moves to the next round (where the same examples or other labeled examples, say D_4, D_5, D_6 , will be used in the same way).

The problem is that sometimes worker nodes can be stragglers [7, 5, 4] *i.e.* delay significantly in computing a gradient. First, we discuss one way to resolve this problem if we replicate some data across machines by considering the placement in Fig.1 (b) but without coding. As can be seen, in Fig. 1 (b) each example is replicated two times using a specific placement policy. Each worker is assigned to compute two gradients on the two examples they have for this round. For example, W_1 will compute vectors g_1 and g_2 . Now let's assume that W_3 is the straggler. If we use control messages, W_1, W_2 can notify the master A that they are done. Subsequently, if feedback is used, the master can ask W_1 to send g_1 and g_2 and W_2 to send g_3 . These feedback control messages can be much smaller than the actual gradient vectors but are still a system complication that can cause delays. However, feedback makes it possible for a centralized node to coordinate the workers. One can reduce network communication significantly by simply asking W_1 to send the sum of two gradient vectors $g_1 + g_2$ instead of sending both. The master can then

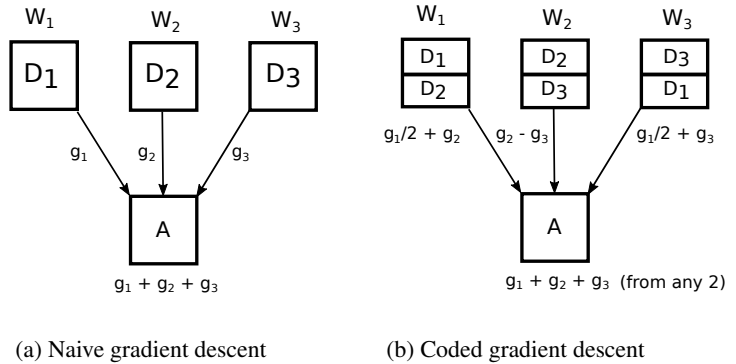


Figure 1

create the global gradient on this batch by summing these two vectors. Unfortunately, which linear combination must be sent depends on who is the straggler: If W_2 was the straggler then W_1 should be sending g_2 and W_3 sending $g_1 + g_3$ so that their sum is the global gradient $g_1 + g_2 + g_3$.

In this paper we show that feedback and coordination is not necessary: every worker can send a *single linear combination of gradient vectors* without knowing who the straggler will be. The main coding theoretic question we investigate is how to design three linear combinations so *that any two* contain the $g_1 + g_2 + g_3$ vector in their span. In our example, in Fig. 1b, W_1 sends $\frac{1}{2}g_1 + g_2$, W_2 sends $g_2 - g_3$ and W_3 sends $\frac{1}{2}g_1 + g_3$. The reader can verify that A can obtain the vector $g_1 + g_2 + g_3$ from *any two out of these three vectors*. For instance, $g_1 + g_2 + g_3 = 2(\frac{1}{2}g_1 + g_2) - (g_2 - g_3)$. We call this idea *gradient coding*.

We consider this problem in the general setting of n machines and **any** s stragglers. We first establish a lower bound: to compute gradients on all the data in the presence of **any** s stragglers, each partition must be replicated $s + 1$ times across machines. We design two placement and gradient coding schemes that match this optimal $s + 1$ replication factor. We further consider a partial straggler setting, wherein we assume that a straggler can compute gradients at a fraction of the speed of others, and show how our scheme can be adapted to such scenarios. All proofs/supplementary material can be found in the appendix.

We compare our scheme with the popular *ignoring the stragglers* approach [1]: simply doing a gradient step when most workers are done. We see that while ignoring the stragglers is faster, this loses some data and which can hurt the generalization error. This can be especially pronounced in supervised learning with unbalanced labels or heavily unbalanced features since a few examples may contain critical, previously unseen information.

1.1 Related Work

The slow machine problem is the Achilles heel of many distributed learning systems that run in modern cloud environments. Recognizing that, some recent work has advocated asynchronous approaches [7, 5, 9] to learning. While asynchronous updates are a valid way to avoid slow machines, they do give up many other desirable properties, including faster convergence rates, amenability to analysis, and ease of reproducibility and debugging.

Attacking the slow machine problem in synchronous machine learning algorithms has surprisingly not received much attention in the literature. There do exist general systems solutions such as speculative execution [11] but we believe that approaches tailored to machine learning can be vastly more efficient. In [1] the authors use synchronous minibatch SGD and request a small number of additional worker machines so that they have an adequate minibatch size even when some machines are slow. However, this approach does not handle well machines that are consistently slow and the data on those machines might never participate in training. In [10] the authors describe an approach for dealing with failed machines by approximating the loss function in the failed partitions with a linear approximation at the last iterate before they failed. Since the linear approximation is only valid at a small neighborhood of the model parameters, this approach can only work if failed data partitions are restored fairly quickly.

The work of [6] is perhaps the closest in spirit to our work, using coding theory and treating stragglers as erasures in the transmission of the computed results. However, we focus on codes for recovering the batch gradient of *any* loss function while [6] describe techniques for mitigating stragglers in two different distributed applications: data shuffling and matrix multiplication. We also mention [8], which investigates a generalized view of the coding ideas in [6].

2 Preliminaries

Given data $\mathbf{D} = \{(x_1, y_1), \dots, (x_d, y_d)\}$, with each tuple $(x, y) \in \mathbb{R}^p \times \mathbb{R}$, several machine learning tasks can be expressed as solving the following problem: $\beta^* = \arg \min_{\beta \in \mathbb{R}^p} \sum_{i=1}^d \ell(\beta; x_i, y_i)$, where $\ell(\cdot)$ is a task-specific loss function. Typically, this optimization problem can be solved using gradient-based approaches. Let $g := \sum_{i=1}^d \nabla \ell(\beta^{(t)}; x_i, y_i)$ be the gradient of the loss at the current model $\beta^{(t)}$. Then the updates are of the form: $\beta^{(t+1)} = h(\beta^{(t)}, g)$, where h is a gradient based optimizer. Several methods such as gradient descent, accelerated gradient, conditional gradient

(Frank-Wolfe), LBFGS, and bundle methods fit in this framework. However, if the number of samples, d , is large, a computational bottleneck in the above update step is the computation of the gradient, g , whose computation can be distributed.

Notation. Throughout this paper, we let d denote the number of samples, n denote the number of workers, k denote the number of data partitions, and s denote the number of stragglers/failures. The n workers are denoted as W_1, W_2, \dots, W_n . The partial gradients over k data partitions are denoted as g_1, g_2, \dots, g_k . The i^{th} row of some matrices A or B is denoted as a_i or b_i respectively. For any vector $\mathbf{x} \in \mathbb{R}^n$, $\text{supp}(\mathbf{x})$ denotes its support *i.e.* $\text{supp}(\mathbf{x}) = \{i \mid \mathbf{x}_i \neq 0\}$. $\|\mathbf{x}\|_0$ denotes its ℓ_0 -norm *i.e.* the cardinality of the support. $\mathbf{1}_{p \times q}$ and $\mathbf{0}_{p \times q}$ denote all 1s and all 0s matrices respectively, with dimension $p \times q$. Finally, for any $r \in \mathbb{N}$, $[r]$ denotes the set $\{1, \dots, r\}$.

2.1 The General Setup

We can generalize the scheme in Figure 1b by setting up a system of linear equations:

$$AB = \mathbf{1}_{f \times k} \quad (1)$$

where f is the number of combinations of surviving workers/non-stragglers, and we have matrices $A \in \mathbb{R}^{f \times n}$, $B \in \mathbb{R}^{n \times k}$.

To relate Eq. (1) to a scheme robust to some failures/stragglers, we say that b_i , the i^{th} row of B , is associated with the i^{th} worker, W_i . The support of b_i , $\text{supp}(b_i)$, corresponds to the data partitions that worker W_i has access to, and the entries of b_i encode a linear combination over their gradients that worker W_i transmits. Let $\bar{g} \in \mathbb{R}^{k \times d}$ be a matrix with each row being the partial gradient of a data partition *i.e.* $\bar{g} = [g_1, g_2, \dots, g_k]^T$. Then, worker W_i transmits $b_i \bar{g}$. Note that to transmit $b_i \bar{g}$, W_i only needs to compute the partial gradients on the partitions in $\text{supp}(b_i)$. Each row of the matrix A corresponds to a specific failure/straggler scenario, to which robustness is desired. In particular, the support of its i^{th} row, $\text{supp}(a_i)$, corresponds to the scenario wherein the workers $\text{supp}(a_i)$ have survived. Then, by construction, we have:

$$a_i B \bar{g} = [1, 1, \dots, 1] \bar{g} = \left(\sum_{j=1}^k g_j \right)^T \quad \text{and} \quad a_i B \bar{g} = \sum_{k \in \text{supp}(a_i)} a_i(k) (b_k \bar{g}) \quad (2)$$

Thus, the entries of a_i encode a linear combination which, when taken over the transmitted gradients of the non-straggler workers, would yield the full gradient.

Going back to the example in Fig. 1b, the corresponding A and B matrices under this generalization are: $A = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & -1 & 0 \end{pmatrix}$ and $B = \begin{pmatrix} 1/2 & 1 & 0 \\ 0 & 1 & -1 \\ 1/2 & 0 & 1 \end{pmatrix}$, with $f = 3, n = 3, k = 3$. It is easy to check that $AB = \mathbf{1}_{3 \times 3}$. In general, we shall seek schemes, through the construction of (A, B) , which are robust to **any** s stragglers.

3 Full Stragglers

In this section, we consider schemes robust to any s stragglers, given n workers (with $s < n$). We assume that a straggler is a *full straggler* *i.e.* it can be arbitrarily slow to the extent of complete failure. We show how to construct the matrices A and B , with $AB = \mathbf{1}$, such that the scheme (A, B) is robust to **any** s stragglers.

Consider any such scheme (A, B) . Since every row of A represents a set of non-straggler workers, all possible sets over $[n]$ of size $(n - s)$ must be supports in the rows of A . Thus $f = \binom{n}{n-s} = \binom{n}{s}$ *i.e.* the total number of ways to choose s stragglers out of n workers. Now, since each row of A represents a linear span over some rows of B , and since $AB = \mathbf{1}$, this leads us to the following requirement on B :

Condition 1 (B-Span). Consider any scheme (A, B) robust to **any** s stragglers, given n workers (with $s < n$). Then we require that for every subset $I \subseteq [n], |I| = n - s$:

$$\mathbf{1}_{1 \times k} \in \text{span}\{b_i \mid i \in I\} \quad (3)$$

where $\text{span}\{\cdot\}$ is the span of vectors. In other words, the all 1s vector must be in the span of any $n - s$ rows of B .

The B-Span condition above is of course necessary. However, it is also sufficient. In particular, given a B satisfying Condition 1, we can construct an A (described in Algorithm 1 in the supplementary) such that $AB = \mathbf{1}$, and A has the support structure discussed above.

Therefore, to obtain a valid scheme (A, B) , we only need to furnish a B satisfying the B-Span condition (Condition 1). A trivial B that works is $B = \mathbf{1}_{n \times k}$, the all ones matrix. However, this is wasteful since it implies that each worker gets all the partitions and computes the full gradient. Our goal is to construct B satisfying Condition 1 while also being as sparse as possible in each row. In this regard, we have the following theorem, which gives a lower bound on the number of non-zeros in any row of B .

Theorem 1 (Lower Bound on B’s density). *Consider any scheme (A, B) robust to **any** s stragglers, given n workers (with $s < n$) and k partitions. Then, if all rows of B have the same number of non-zeros, we must have: $\|b_i\|_0 \geq \frac{k}{n}(s + 1)$ for any $i \in [n]$.*

Theorem 1 implies that any scheme (A, B) that assigns the same amount of data to all the workers must assign at least $\frac{s+1}{n}$ fraction of the data to each worker. Since this fraction is independent of k , for the remainder of this paper we shall assume that $k = n$ i.e. the number of partitions is the same as the number of workers. In this case, B must be a square matrix satisfying Condition 1, with each row having atleast $(s + 1)$ non-zeros. In the sequel, we demonstrate two constructions for B which achieve this lower bound.

3.1 Fractional Repetition Scheme

We now provide a construction for B that works by replicating the task done by a subset of the workers. We note that this construction is only applicable when the number of workers, n , is a multiple of $(s + 1)$, where s is the number of stragglers we seek robustness to. In this case, the construction is as follows:

- We divide the n workers into $(s + 1)$ groups of size $(n/(s + 1))$.
- In a group, we divide the data equally and disjointly, assigning $(s + 1)$ partitions to each worker
- All the groups are replicas of each other
- When finished computing, every worker transmits the sum of its partial gradients

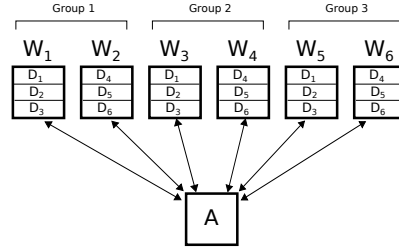


Figure 2: Fractional Repetition Scheme for $n = 6, s = 2$

Fig. 2 shows an instance of the above construction for $n = 6, s = 2$. A general description of B constructed in this way (denoted as B_{frac}) is shown in Eq. (6) (in the supplementary). It is easy to see that this construction can yield robustness to s stragglers. Since any particular partition is replicated over $(s + 1)$ workers, any s stragglers leaves at least one non-straggler worker to process it. Now, we have the following theorem.

Theorem 2. *Consider B_{frac} constructed as in Eq. (6), for a given number of workers n and stragglers $s(< n)$. Then, B_{frac} satisfies the B-Span condition (Condition 1). Consequently, the scheme (A, B_{frac}) , with A constructed using Algorithm 1, is robust to any s stragglers.*

The construction of B_{frac} matches the density lower bound in Theorem 1. Also, the above theorem shows that the scheme (A, B_{frac}) , with A constructed from Algorithm 1, is robust to s stragglers.

3.2 Cyclic Repetition Scheme

In this section, we construct an alternate B which also matches the lower bound in Theorem 1 and satisfies Condition 1. However, in contrast to the previous section, this construction does not require n to be divisible by $(s + 1)$. Here, instead of assigning disjoint collections of partitions, we consider a cyclic assignment of $(s + 1)$ partitions to the workers. We construct a $B = B_{cyc}$ with the support

structure:

$$\text{supp}(B_{cyc}) = \begin{bmatrix} \star & \star & \cdots & \star & \star & 0 & 0 & \cdots & 0 & 0 \\ 0 & \star & \star & \cdots & \star & \star & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \star & \star & \cdots & \star & \star \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \star & \cdots & \star & \star & 0 & 0 & \cdots & 0 & 0 & \star \end{bmatrix}_{n \times n} \quad (4)$$

where \star indicates non-zero entries in B_{cyc} . The first row of B_{cyc} has its first $(s + 1)$ entries assigned as non-zero. As we move down the rows, the positions of the $(s + 1)$ non-zero entries shift one step to the right, and cycle around until the last row.

Given the support structure in Eq. 4, the actual non-zero entries must be carefully assigned in order to satisfy Condition 1. The basic idea is to pick every row of B_{cyc} , with the fixed support, to lie in a suitable subspace S that contains the all ones vector $\mathbf{1}_{n \times 1}$. It turns out that a choice of random subspace S (described as the null space of a random matrix satisfying the so-called MDS property in coding theory) can produce such a B_{cyc} . Algorithm 2 (in the supplementary) describes the construction of B_{cyc} . Then, we have the following theorem.

Theorem 3. Consider a B_{cyc} constructed using the randomized construction in Algorithm 2, for a given number of workers n and stragglers $s (< n)$. Then, with probability 1, B_{cyc} satisfies the B-Span condition (Condition 1). Consequently, the scheme (A, B_{cyc}) , with A constructed using Algorithm 1, is robust to any s stragglers.

4 Partial Stragglers

In this section, we review our earlier assumption of *full stragglers*. Instead, we consider a more plausible scenario of slow workers, but assume a known slowdown factor. We say that a straggler is an α -*partial straggler* (with $\alpha > 1$) if it is at most α slower than any non-straggler. This means that if a non-straggler completes a task in time T , an α -*partial straggler* would require at most αT time to complete it. Now, we augment our distribution schemes to be robust to **any** s stragglers, assuming any straggler is an α -*partial straggler*.

Note that our earlier constructions are still applicable: a scheme (A, B) , with $B = B_{frac}$ or $B = B_{cyc}$, would still provide robustness to s partial stragglers. However, given that no machine is slower than a factor of α , a more efficient scheme is possible by exploiting at least some computation on every machine. Our basic idea is to couple our earlier schemes with a naive distribution scheme, but on different parts of the data. We split the data into a *naive* component, and a *coded* component. The key is to do the split such that whenever an α -partial straggler is done processing its *naive* partitions, a non-straggler would be done processing both its *naive* and *coded* partitions. In general, for any (n, s, α) , our two-stage scheme works as follows:

- We split the data \mathbf{D} into $n + n \frac{s+1}{\alpha-1}$ equal-sized partitions — of which n partitions are *coded* components, and the rest are *naive* components
- Each worker gets $\frac{s+1}{\alpha-1}$ *naive* partitions, distributed disjointly.
- Each worker gets $(s + 1)$ *coded* partitions, distributed according to an (A, B) distribution scheme robust to s stragglers (e.g. with $B = B_{frac}$ or $B = B_{cyc}$)
- Any worker, W_i , first processes all its *naive* partitions and sends the sum of their gradients to the aggregator. It then processes its *coded* partitions, and sends a linear combination, as per the (A, B) distribution scheme.

Note that each worker has to send two partial gradients, instead on one, as in earlier schemes. However, a speedup gained in processing a smaller fraction of the data may mitigate this overhead

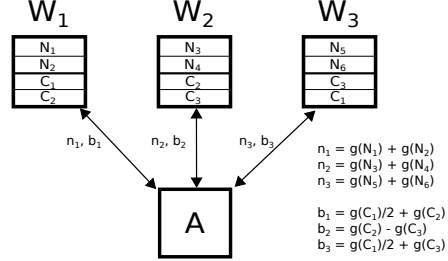


Figure 3: Scheme for Partial Stragglers, $n = 3, s = 1, \alpha = 2$. $g(\cdot)$ represents the partial gradient.

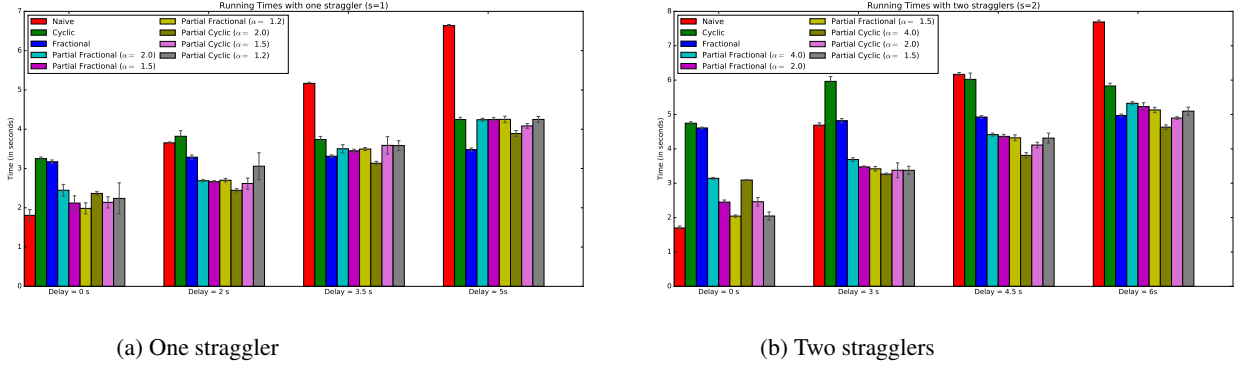


Figure 4: Empirical running time on EC2 with $n = 12$ machines for one and two stragglers. The other machines run at normal speed while the stragglers are artificially delayed. As expected, the baseline naive scheme that waits for the stragglers has very poor performance as the delay increases. *Cyclic* and *Fractional* are designed for one straggler in the left plot and two in the right. Therefore, we theoretically expect that these two schemes would not be influenced at all by the delay of the straggler. Still due to several system implementation reasons we observe some slight delays. The partial schemes are designed for some machines running at full speed and the stragglers running at an α slowdown rate. For example, $\alpha = 2$ on the left plot means that one straggler processes one example in the time it takes for normal workers to process two. Interestingly, we observe that partial schemes have good robustness even when operated outside their design range. We note that the partial straggler schemes for $\alpha = 1.2$ need to only replicate approximately 10% of the blocks.

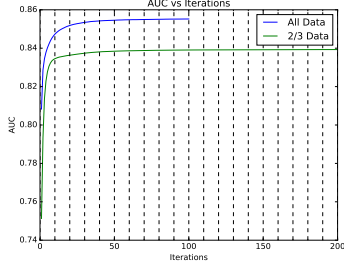
in communication, since now each non-straggler has to process an $\frac{s+1}{n} \left(\frac{\alpha}{s+\alpha} \right)$ fraction of the data, as opposed to an $\frac{s+1}{n}$ fraction in earlier schemes. Fig. 3 illustrates our two-stage strategy for $n = 3, s = 1, \alpha = 2$. We see that each non-straggler gets $4/9 = 0.44$ fraction of the data, instead of a $2/3 = 0.67$ fraction (for e.g. in Fig 1b).

5 Experiments

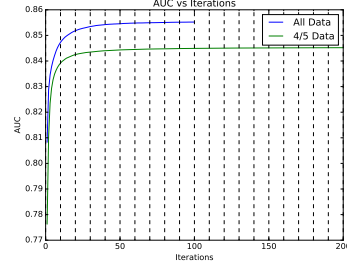
In this section, we present experimental results comparing the proposed gradient coding schemes with baseline approaches. We implemented the distributed schemes in python using MPI4py[2], an open source MPI implementation. Our experiments were performed on a cluster of $n = 12$ machines (m1.small instances) on Amazon EC2. We compare against the *naive* scheme where the data is divided uniformly across all the workers without replication.

Artificial Dataset: In our first experiment we solve a logistic regression problem using gradient descent. We artificially generate a dataset of $d = 554400$ samples $\mathbf{D} = \{(x_1, y_1), \dots, (x_d, y_d)\}$, from the model $x \sim 0.5 \times \mathcal{N}(\mu_1, I) + 0.5 \times \mathcal{N}(\mu_2, I)$ (for random $\mu_1, \mu_2 \in \mathbb{R}^p$), and $y \sim \text{Ber}(p)$, with $p = 1/(\exp(2x^T \beta^*) + 1)$, where $\beta^* \in \mathbb{R}^p$ is the true regressor. In our experiments, we use a model dimension of $p = 100$, and chose β^* randomly. Based on the scheme being considered, each worker loads a certain number of partitions of the data into memory before starting the iterations. In iteration t the aggregator sends the latest model $\beta^{(t)}$ to all the workers (using `isend()`). Each worker receives the model (using `irecv()`) and starts a gradient computation. Once finished, it sends its gradient(s) back to the aggregator. When sufficiently many workers have returned with their gradients, the aggregator computes the overall gradient, performs a descent step, and moves on to the next iteration. We artificially add delays to s random workers in each iteration (using `time.sleep()`). Figure 4 presents the results of our experiments with $s = 1$ and $s = 2$ stragglers. Note that for partial straggler schemes, α denotes the *slowness* factor.

Real Dataset: We also compare our approach against the scheme that *ignores the stragglers*. In this case we use a real dataset for binary classification and train a logistic regression models. The baseline scheme waits for the first $n - s$ machines to compute gradients and ignores the rest. In other words, ignoring the s stragglers corresponds to batch SGD using a $(n - s)/n$ fraction of the data. On the contrary, our scheme provides fault tolerance since data blocks are replicated across $s + 1$ machines and we do not need to wait for the stragglers to finish. Therefore we can perform batch SGD with each batch involving data from all the machines, or even full *gradient descent*. Figure 5 shows a comparison of generalization error (using the AUC metric) for three and five machines and one straggler. As can be seen, our framework tolerates $s = 1$ straggler so it is a factor of two slower compared to ignoring the stragglers. However, it will see all the data while the scheme that



(a) Missing 1/3 of the data



(b) Missing 1/5 of the data

Figure 5: Generalization Error comparison of our scheme with *ignoring stragglers*

ignores stragglers misses some data ($1/3$ and $1/5$ respectively for 3 and 5 machines). Note that in this experiment the straggling worker was fixed to be the same machine across iterations.

We see from Fig. 5a and 5b that ignoring stragglers can allow faster iterations but can increase the generalization error since some examples may be lost. As the number of machines increases, this effect should not be as pronounced if the number of stragglers is scaling as a constant fraction of the number of machines in the cluster. If however, we expect a fraction, e.g. 5% of the machines to be significantly slower than the rest, ignoring 5% of the examples could have an effect in learning. This can be particularly problematic for datasets with labels or features that are very unbalanced: for those cases missing even a few important examples can drastically increase the generalization error.

References

- [1] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting Distributed Synchronous SGD. *ArXiv e-prints*, April 2016.
- [2] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. New Computational Methods and Software Tools.
- [3] S. H. Dau, W. Song, Z. Dong, and C. Yuen. Balanced sparsest generator matrices for mds codes. In *2013 IEEE International Symposium on Information Theory*, pages 1889–1893, July 2013.
- [4] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc’ Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*. 2012.
- [5] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*, 2013.
- [6] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris S. Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *CoRR*, abs/1512.02673, 2015.
- [7] Mu Li, David G Andersen, Alex J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.
- [8] Songze Li, Mohammad Ali Maddah-Ali, and Amir Salman Avestimehr. A unified coding framework for distributed computing with straggling servers. *CoRR*, abs/1609.01690, 2016.
- [9] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. *CoRR*, abs/1605.09774, 2016.
- [10] Shravan Narayanamurthy, Markus Weimer, Dhruv Mahajan, Tyson Condie, Sundararajan Sellamanickam, and S. Sathiya Keerthi. Towards resource-elastic machine learning, 2013.
- [11] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, volume 8, page 7, 2008.

6 Appendix - Computing A

Algorithm 1 Algorithm to compute A

Input : B satisfying Condition 1, $s(< n)$

Output : A such that $AB = \mathbf{1}_{\binom{n}{s} \times n}$

```

 $f = \text{binom}(n, s);$ 
 $A = \text{zeros}(f, n);$ 
foreach  $I \subseteq [n]$  s.t.  $|I| = (n - s)$  do
     $a = \text{zeros}(1, k);$ 
     $x = \text{ones}(1, k) / B(I, :);$ 
     $a(I) = x;$ 
     $A = [A; a];$ 
end

```

We have the following lemma.

Lemma 1. Consider $B \in \mathbb{R}^{n \times k}$ satisfying Condition 1 for some $s < n$. Then, Algorithm 1, with input B and s , yields an $A \in \mathbb{R}^{\binom{n}{s} \times n}$ such that $AB = \mathbf{1}_{\binom{n}{s} \times n}$ and the scheme (A, B) is robust to any s full stragglers.

7 Appendix - Constructing B: Fractional Repetition Scheme

Each group of workers in this scheme can be denoted by a block matrix $\bar{B}_{\text{block}}(n, s) \in \mathbb{R}^{\frac{n}{s+1} \times n}$. We define:

$$\bar{B}_{\text{block}}(n, s) = \begin{bmatrix} \mathbf{1}_{1 \times (s+1)} & \mathbf{0}_{1 \times (s+1)} & \cdots & \mathbf{0}_{1 \times (s+1)} \\ \mathbf{0}_{1 \times (s+1)} & \mathbf{1}_{1 \times (s+1)} & \cdots & \mathbf{0}_{1 \times (s+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{1 \times (s+1)} & \mathbf{0}_{1 \times (s+1)} & \cdots & \mathbf{1}_{1 \times (s+1)} \end{bmatrix}_{\frac{n}{s+1} \times n} \quad (5)$$

Thus, the first worker in the group gets the first $(s + 1)$ partitions, the second worker gets the second $(s + 1)$ partitions, and so on. The overall B is simply $(s + 1)$ replicated copies of $\bar{B}_{\text{block}}(n, s)$:

$$B = B_{\text{frac}} = \begin{bmatrix} \bar{B}_{\text{block}}^{(1)} \\ \bar{B}_{\text{block}}^{(2)} \\ \vdots \\ \bar{B}_{\text{block}}^{(s+1)} \end{bmatrix}_{n \times n} \quad (6)$$

where for each $t \in \{1, \dots, s + 1\}$, $\bar{B}_{\text{block}}^{(t)} = \bar{B}_{\text{block}}(n, s)$.

8 Appendix - Constructing B: Cyclic Repetition Scheme

Algorithm 2 Algorithm to construct $B = B_{\text{cyc}}$ under the Cyclic repetition scheme

Input : $n, s(< n)$

Output : $B \in \mathbb{R}^{n \times n}$ with $(s + 1)$ non-zeros in each row

```

 $H = \text{randn}(s, n);$ 
 $H(:, n) = -\text{sum}(H(:, 1 : n - 1), 2);$ 
 $B = \text{zeros}(n);$ 
for  $i = 1 : n$  do
     $j = \text{mod}(i - 1 : s + i - 1, n) + 1;$ 
     $B(i, j) = [1; -H(:, j(2 : s + 1)) \setminus H(:, j(1))];$ 
end

```

Given the support structure in Eq. 4, the actual non-zero entries must be carefully assigned in order to satisfy Condition 1. The basic idea is to pick every row of B_{cyc} , with the fixed support, to lie in a suitable subspace S that contains the all ones vector $\mathbf{1}_{n \times 1}$. Then, given any $(n - s)$ rows of B_{cyc} , we show that their linear span also contains $\mathbf{1}_{n \times 1}$. Consider a $(n - s)$ dimensional subspace, $S = \{x \in \mathbb{R}^n \mid Hx = 0, H \in \mathbb{R}^{s \times n}\}$, where the null space of H describes the subspace S . Then, to make the rows of B_{cyc} lie in S , we require the null space of H to contain vectors with different supports as in Eq. 4. This turns out to be equivalent to requiring that any s columns of H are linearly independent, and is also referred to as the MDS property in coding theory. Based on a random choice of such an H , we are able to construct a B_{cyc} overall with the support structure in Eq. 4, and which satisfies Condition 1. Algorithm 2 describes the construction of B_{cyc} .

9 Appendix - Proofs

9.1 Proof of Lemma 1

By Condition 1, we know that for any $I \subseteq [n]$, $|I| = n - s$, we have $\mathbf{1} \in \text{span}\{b_i \mid i \in I\}$. In other words, there exists atleast one $x \in \mathbb{R}^{(n-s)}$ such that:

$$xB(I, :) = \mathbf{1} \quad (7)$$

Therefore, by construction, we have: $AB = \mathbf{1}_{\binom{n}{s} \times n}$, and the scheme (A, B) is robust to **any** s stragglers.

9.2 Proof of Theorem 1

Consider any scheme (A, B) robust to **any** s stragglers, with $B \in \mathbb{R}^{n \times k}$. Now, construct a bipartite graph between n workers, $\{W_1, \dots, W_n\}$, and k partitions, $\{P_1, \dots, P_k\}$, where we add an edge (i, j) if worker i and partition j is worker i has access to partition j . In other words, for any $i \in [n], j \in [k]$:

$$e_{ij} = \begin{cases} 1 & \text{if } B(i, j) \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

Now, it is easy to see that the degree of the i^{th} worker W_i is $\|b_i\|_0$.

Also, for any partition P_j , its degree must be at least $(s + 1)$. If its degree is s or less, then consider the scenario where all its neighbors are stragglers. In this case, there is no non-straggler worker with access to P_j , which contradicts robustness to **any** s stragglers.

Based on the above discussion, and using the fact that the sum of degrees of the workers in the bipartite graph must be the same as the sum of degrees of partitions, we get:

$$\sum_{i=1}^n \|b_i\|_0 \geq k(s + 1) \quad (9)$$

Since we assume all workers get access to the same number of partitions, this gives:

$$\|b_i\|_0 \geq \frac{k(s + 1)}{n}, \text{ for any } i \in [n] \quad (10)$$

9.3 Proof of Theorem 2

Consider groups of partitions $\{G_1, \dots, G_{n/(s+1)}\}$ as follows:

$$\begin{aligned} G_1 &= \{P_1, \dots, P_{s+1}\} \\ G_2 &= \{P_{s+2}, \dots, P_{2s+2}\} \\ &\vdots \\ G_{n/(s+1)} &= \{P_{n-s}, \dots, P_n\} \end{aligned} \quad (11)$$

Fix some set $I \subseteq [n]$, $|I| = n - s$. Based on our construction, it is easy to observe that for any group G_j , there exists some index in I , say $i_{G_j} \in I$, such that the corresponding row in B , $b_{i_{G_j}}$ has all 1s

at partitions in G_j and 0s elsewhere. This is because there are $(s + 1)$ rows of B that correspond in this way to G_j (one in each block $\overline{B}_{\text{block}}$), and so atleast one would survive in the set I of cardinality $(n - s)$. Now, it is trivial to see that:

$$\mathbf{1} \in \text{span}\{b_{i_{G_j}} \mid j = 1, \dots, n/(s + 1)\} \quad (13)$$

Also, since

$$\text{span}\{b_{i_{G_j}} \mid j = 1, \dots, n/(s + 1)\} \subseteq \text{span}\{b_i \mid i \in I\}, \quad (14)$$

we have $\mathbf{1} \in \text{span}\{b_i \mid i \in I\}$.

Finally, since the above holds for any set I , we get that B satisfies Condition 1. The remainder of the theorem follows from Lemma 1.

9.4 Proof of Theorem 3

Consider the subspace given by the null space of the random matrix H (constructed in Algorithm 2):

$$S = \{x \in \mathbb{R}^n \mid Hx = 0\} \quad (15)$$

Note that H has $(n - 1)s$ different random values (s for each column), since its last column is simply the negative sum of its previous $(n - 1)$ columns. Now, we have the following Lemma listing some properties of H and S .

Lemma 2. *Consider $H \in \mathbb{R}^{s \times n}$ as constructed in Algorithm 2, and the subspace S as defined in Eq. 15. Then, the following hold:*

- Any s columns of H are linearly independent with probability 1
- $\dim(S) = n - s$ with probability 1
- $\mathbf{1} \in S$, where $\mathbf{1}$ is the all-ones vector

For $i \in [n]$, let S_i denote the set $S_i = \{i \bmod n, (i + 1) \bmod n, \dots, (i + s) \bmod n\}$. Then, S_i corresponds to the support of the i^{th} row of B in our construction, as also given by the support structure in Eq. (4).

Recall that we denote the i^{th} row of B by b_i . By our construction, we have:

$$\begin{aligned} b_i(i) &= 1 \\ b_i(S_i \setminus \{i\}) &= -H_{S_i \setminus \{i\}}^{-1} H_i \end{aligned} \quad (16)$$

Now, we have the following lemma;

Lemma 3. *Consider the i^{th} row of B constructed using Algorithm 2 (also shown in Eq. 16). Then,*

- $b_i \in S$
- Every element of $b_i(S_i \setminus \{i\})$ is non-zero with probability 1
- For any subset $I \subseteq [n]$, $|I| = n - s$, the set of vectors $\{b_i \mid i \in I\}$ is linearly independent with probability 1

Now, using Lemma 3, we can conclude that for any subset $I \subseteq [n]$, $|I| = n - s$, $\dim(\text{span}\{b_i \mid i \in I\}) = n - s$ and $\text{span}\{b_i \mid i \in I\} \subseteq S$. Consequently, from Lemma 2, since $\dim(S) = n - s$ and $\mathbf{1} \in S$, this implies that:

$$\text{span}\{b_i \mid i \in I\} = S \text{ with probability 1} \quad (17)$$

and, $\mathbf{1} \in \text{span}\{b_i \mid i \in I\}$. Taking union bound over every I shows that B satisfies Condition 1. The remainder of the theorem follows from Lemma 1.

9.4.1 Proof of Lemma 2

Consider any subset $I \subseteq [n]$, $|I| = s$ such that $n \notin I$. Then, all the elements of H_I are independent, and $\det(H_I)$ is a polynomial in the elements of H_I . Consequently, since every element is drawn from a continuous probability distribution (in particular, gaussian), the set $\{H_I \mid \det(H_I) = 0\}$ is a zero measure set. So, $P(\det(H_I) \neq 0) = 1$, and thus the columns of H_I are linearly independent with probability 1.

If $n \in I$, then we have:

$$\det(H_I) = \det(\tilde{H}) \quad (18)$$

where we let $\tilde{H} = [H_{I \setminus \{n\}}, -\sum_{i \in [n] \setminus I} H_i]$. The elements of \tilde{H} are independent, so using the same argument as above, we again have $P(\det(H_I) = \det(\tilde{H}) \neq 0) = 1$. Finally, taking a union bound over all sets I of cardinality s shows that any s columns of H are linearly independent.

Since any s columns in H are linearly independent, this implies that $\text{rank}(H) = s$. Since the subspace S is simply the null space of H , we have $\dim(S) = n - s$.

Finally, since $H_n = -\sum_{i \in [n-1]} H_i$ (by construction), we have $H\mathbf{1} = 0$ and thus $\mathbf{1} \in S$.

9.4.2 Proof of Lemma 3

By construction of b_i , we have:

$$Hb_i = H_i + H_{S_i \setminus \{i\}} b_i(S_i \setminus \{i\}) = H_i - H_i = 0 \quad (19)$$

Thus, $b_i \in S$.

Now, if possible, let for some $k \in S_i \setminus \{i\}$, $b_i(k) = 0$. Then, since $b_i \in S$, we have:

$$Hb_i = H_i + H_{S_i \setminus \{i, k\}} b_i(S_i \setminus \{i, k\}) = 0 \quad (20)$$

Consequently, the set of columns $\{j \mid j \in S_i \setminus \{i, k\}\} \cup \{i\}$ is linearly dependent which contradicts H having any s columns being linearly independent (in Lemma 2). Therefore, we must have every element of $b_i(S_i \setminus \{i\})$ being non-zero.

Now, consider any subset $I \subseteq [n]$, $|I| = n - s$. We shall show that the matrix B_I (corresponding to the rows of B with indices in I) has rank $n - s$ with probability 1. Consequently, the set of vectors $\{b_i \mid i \in I\}$ would be linearly independent. To show this, we consider some $n - s$ columns of B_I , say given by the set $J \subseteq [n]$, $|J| = n - s$, and denote the sub-matrix of columns by $B_{I,J}$. Then, it suffices to show that $\det(B_{I,J}) \neq 0$. Now, by the construction in Algorithm 2, we have: $\det(B_{I,J}) = \text{poly}_1(H)/\text{poly}_2(H)$, for some polynomials $\text{poly}_1(\cdot)$ and $\text{poly}_2(\cdot)$ in the entries of H . Therefore, if we can show that there exists at least one H' with $H'\mathbf{1} = \mathbf{0}$ and $\text{poly}_1(H')/\text{poly}_2(H') \neq 0$, then under a choice of i.i.d. standard gaussian entries of H , we would have:

$$\mathbb{P}(\text{poly}_1(H)/\text{poly}_2(H) \neq 0) = 1 \quad (21)$$

The remainder of this proof is dedicated to showing that such an H' exists. To show this, we shall consider a matrix $\tilde{B} \in \mathbb{R}^{n-s \times n}$ such that $\text{supp}(\tilde{B}) = \text{supp}(B_I)$ and $\det(\tilde{B}_{:,J}) \neq 0$, where $\tilde{B}_{:,J}$ corresponds to the sub-matrix of \tilde{B} with columns in the set J . Given such a \tilde{B} , we shall show that there exists an $s \times n$ matrix H' (with $H'\mathbf{1} = \mathbf{0}$) such that when we run Algorithm 2 with this H' , we get a matrix B' s.t. $B'_I = \tilde{B}$ i.e. the output matrix from Algorithm 2 is identical to our random choice \tilde{B} on the rows in the set I . This suffices to show the existence of an H' such that $\text{poly}_1(H')/\text{poly}_2(H') \neq 0$, since $\text{poly}_1(H')/\text{poly}_2(H') = \det(B'_{I,J}) = \det(\tilde{B}_J) \neq 0$.

Let us pick a random matrix \tilde{B} as:

$$\tilde{B} = B_I^r D \quad (22)$$

where B_I^r is a matrix with the same support as B_I and with each non-zero entry i.i.d. standard gaussian, and D is a diagonal matrix such that $D_{ii} = \sum_{j=1}^{n-s} B_I^r(j, i)$, $i \in [n]$. Note that a consequence of the above choice of \tilde{B} is that the sum of all its rows is the all 1s vector. Now, it can be shown that any $(n - s)$ columns of \tilde{B} form an invertible sub-matrix with probability 1. Let S_i be the support of

the i^{th} row of B . The rows of B_I^r have the supports $S_i, i \in I$. Now because of the cyclic support structure in B , any collection $\{i_1, i_2, \dots, i_k\} (0 \leq k \leq n-s)$ satisfies the property:

$$|\cup_{j=1}^k S_{i_j}| \geq s + k \quad (23)$$

Using Lemma 4 in [3], this implies that there is a perfect matching between the rows of B_I^r and any of its $(n-s)$ columns. Consequently, with probability 1, any $(n-s)$ columns of B_I^r form an invertible sub-matrix. Also, since every column of B_I^r contains atleast one non-zero (again, owing to the support structure of B), this implies that with probability 1, all the diagonal entries of D are non-zero. Combining the above two observations, we can infer that any $(n-s)$ columns of \tilde{B} form an invertible sub-matrix with probability 1.

So far, we have shown existence of a matrix \tilde{B} with the following properties: (i) \tilde{B} has the same support structure as B_I , (ii) any $(n-s)$ columns of \tilde{B} form invertible sub-matrix, (iii) the sum of all rows of \tilde{B} is the all 1s vector. Now, for any such \tilde{B} , we shall show that there exists an H' such that $H' \tilde{B}^T = \mathbf{0}$ such that any s columns of H' form an invertible sub-matrix. This implies that when we run Algorithm 2 with this H' , the output matrix would be the same as \tilde{B} on the rows in the set I . The remainder of the proof then follows from our earlier discussion.

Now, consider any set $Q \subseteq [n], |Q| \leq s$. Suppose we pick any invertible $H'_{:,Q}$, and set $H'_{:, [n] \setminus Q} = -H'_{:,Q} \tilde{B}_{:,Q}^T (\tilde{B}_{:, [n] \setminus Q}^T)^{-1}$. Then, such an H' satisfies $H' \tilde{B}^T = 0$ and its columns in the set Q form an invertible sub-matrix. Now, since invertibility on the set Q simply corresponds to $\det(H'_{:,Q}) \neq 0$ (*i.e.* some fixed polynomial being non-zero), if we actually picked a uniformly random H' on the subspace $H' \tilde{B}^T = 0$, then

$$\mathbb{P} \left(\det(H'_{:,Q}) \neq 0 \mid H' \tilde{B}^T = 0 \right) = 1 \quad (24)$$

Taking a union bound over all Q s, we get that

$$\mathbb{P} \left(\text{any } s \text{ columns of } H' \text{ form an invertible sub-matrix} \mid H' \tilde{B}^T = 0 \right) = 1 \quad (25)$$

Thus, there exists an H' satisfying $H' \tilde{B}^T = 0$ with any s of its columns forming an invertible sub-matrix. Also, since the sum of all rows of \tilde{B} is $\mathbf{1}$, this implies $H' \mathbf{1} = \mathbf{0}$.